Performance Evaluation of MongoDB I/O access patterns

Abdulqawi Saif, Lucas Nussbaum, Ye-Qiong Song

Performance Evaluation of *MongoDB* I/O access patterns

Abdulqawi Saif[1,2,3,4]
abdulqawi.saif@loria.fr

Lucas Nussbaum[1,2,3]
lucas.nussbaum@loria.fr

Ye-Qiong Song[1,2,3]
ye-qion.song@loria.fr

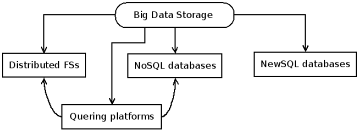
[1]Inria, Villers-les-Nancy, F-54600, France
[2]Université de Lorraine, LORIA, F-54500, France
[3]CNRS, LORIA - UMR 7503, F-54500, France
[4]Xilopix SAS, F-88000, France

September 07, 2017
CLOUD DAYS'2017 - Nancy, France

Context

NoSQL databases

- Principle category of Big Data Storage
- Overcome the scalability issues of SQL DBs
- Almost Cloud-friend systems
- Different architectures and data-models



MongoDB

- A document-based NoSQL database (JSON documents)
- It uses *sharding* for distributing data
- Different tools for easy-deployment and migration (Ex. mongorestore, mongodump)
- Simple to use and fit well with various kinds of apps.

Context

Querying data

- NoSQL flexibility: store data; think how to use after!
- No schema $\Rightarrow$ not only one way to query the data

frequent need to create indexes on demand, on large-scale data!

Context

Querying data

- NoSQL flexibility: store data; think how to use after!
- No schema $\Rightarrow$ not only one way to query the data

frequent need to create indexes on demand, on large-scale data!

Problem statement

- MongoDB takes unjustified amount of time for making indexes on a pre-existed data

this problem could affect other NoSQL databases!

Context

Benchmarking?

⇒ Tools such as *Yahoo! Cloud Serving Benchmarks* (*YCSB*) could:

- report high level metrics (Ex. throughput, latency, ...) ✓
- reduce the effort for comparing different systems ✓

But

- could not reflect the complexity of production-like data ☹
- are not suitable for explaining performance issues ☹

Research objectives

- Identifying the performance issue(s) of MongoDB while indexing data
- Introducing new experimental tools for explaining the causes of performance issue(s)

Experiments

Datasets

Dataset type	(min, avrg, max) in Kb	N. of data units	Size (Gb)
SDU_1	(1, 3, 6)	20,000,000 docs	71

- Every doc x (int, date, 2 x string[min, max], array[1..4] x string[min, max])

MongoDB's storage engine: Wiredtiger

- The default storage engine for persistent data.
- $\_id_s$ for holding the collection's records
- collection is a separate file
- Btree data structure on disks
- Contiguous allocation on disk, but unspanned
- Default page size is 4 KB

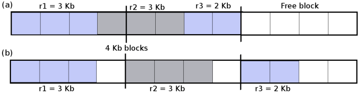


FIGURE – a) Spanned & b) Unspanned allocation

Experiments

Experimental setup

- Experiments are performed on Grid'5000 testbed
  `Machines x 2 CPUs Intel Xeon E5-2630 v3, 8 cores/CPU, 128GB RAM, 2x558GB HDD, 10Gbps ethernet`
- Ubuntu 14.04, Linux 4.9.13, MongoDB v3.4
- wiredTiger cache = 2Gb, replicas are disabled, sharding key is $\_id_s$
- Considering only hash sharding in this talk ⇒ load balancing

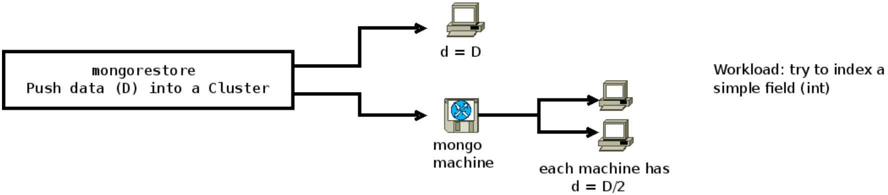
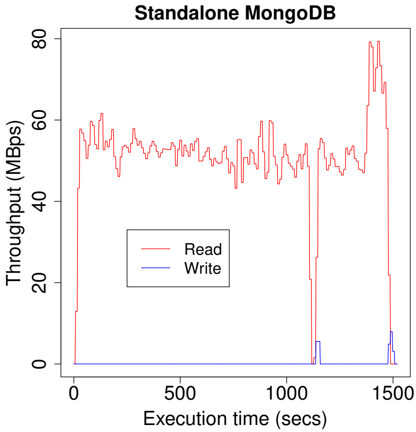
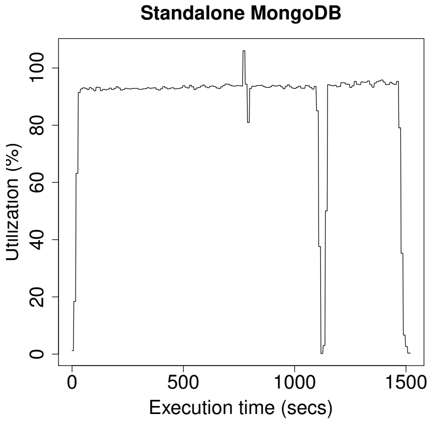


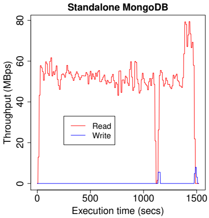
FIGURE – Experimental scenarios

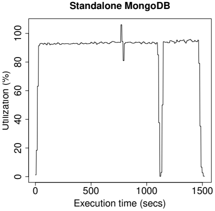
Experiments : Performance results with SDU_1



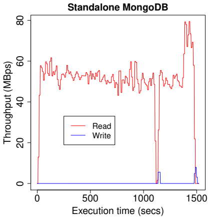


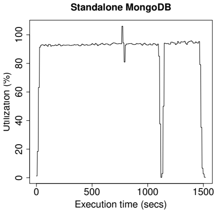
Experiments : Performance results with SDU_1

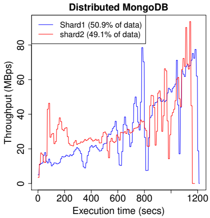


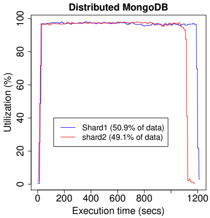


Experiments : Performance results with SDU_1

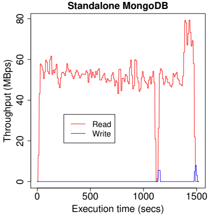


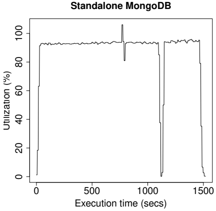


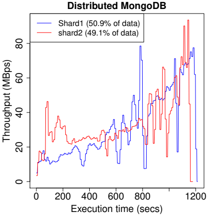


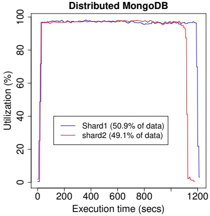


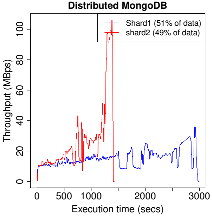
Experiments : Performance results with SDU_1

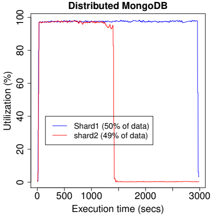




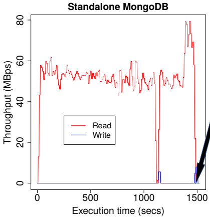


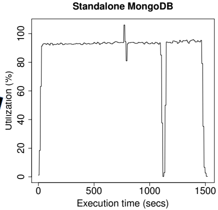


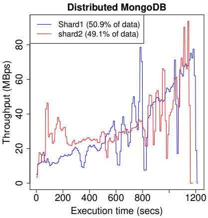


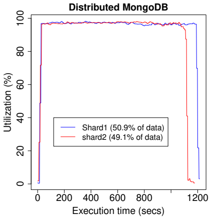


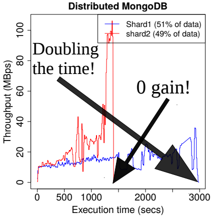
- $\Rightarrow$ Indexing is a 100% read workload
- $\Rightarrow$ I/O bounded operations
- $\Rightarrow$ Data distribution issue
- $\Rightarrow$ Benchmarking is not sufficient to look beyond!

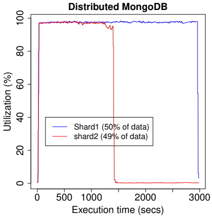




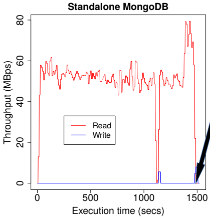


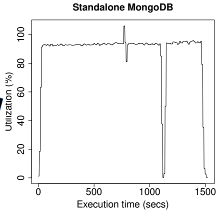


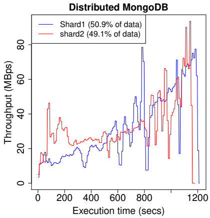


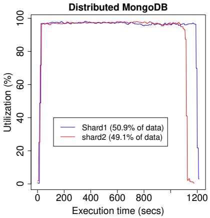


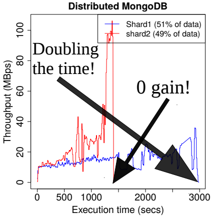
- ⇒ Indexing is a 100% read workload
- ⇒ I/O bounded operations
- ⇒ Data distribution issue
- ⇒ Benchmarking is not sufficient to look beyond!

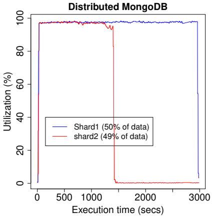












- $\Rightarrow$ Indexing is a 100% read workload
- $\Rightarrow$ I/O bounded operations
- $\Rightarrow$ Data distribution issue
- $\Rightarrow$ Benchmarking is not sufficient to look beyond!

$\Rightarrow$ How to get the main reason behind these results ?

extended Berkeley Packet Filter (eBPF)

- Connect to all Linux data sources : (Kprobes, Uprobes, tracepoints, ...)
- Almost no overhead (4 ns per syscall) [A. Starovoitov]
- In-kernel verifier, JIT-compilation, mapping (kernel-userspace exchange)
- Processing, tracing, filtering inside Linux kernel.
- Suitable to work with system in-production & real data.

BPF compiler collection (BCC)

- Like-standard frontend project for eBPF tools
- No more byte code! ⇒ write your BPF code in restricted C
- Write your frontend code using Python or Lua
- Towards script-driven tracing

Dynamic tracing with eBPF, our tool

An eBPF tool is built to evaluate the I/O patterns

- I/O access are no longer considered as black boxes !
- A generic tool, works on VFS layer
- It traces all I/O read requests
- It reports all files offsets, req. latencies & data size
- Could filter the results by file

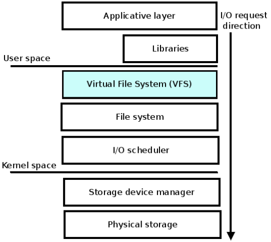
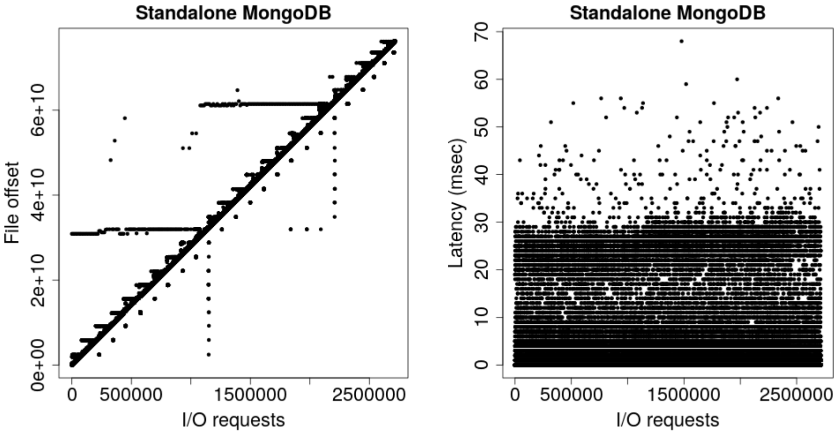


FIGURE – Linux abstracted I/O stack

Tracing impact on our experiments

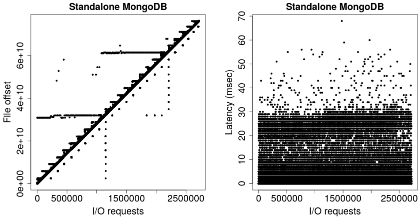
- Overhead cost is lower than 0.8% of execution time in all Experiments. ☺
- Basically, it depends on the number issued I/O requests.

Experiments : eBPF results with SDU_1

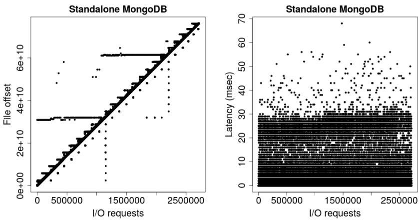


I/O access pattern and latency on standalone config.

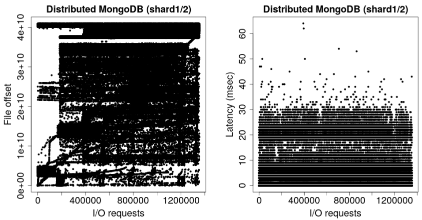
Experiments : eBPF results with SDU_1



I/O access pattern and latency on standalone config.

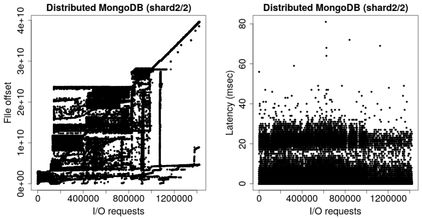


I/O access pattern and latency on standalone config.



First shard results in two-shards config (50.9% of data)

- $\Rightarrow$ Acceptable access pattern on standalone
- $\Rightarrow$ Almost random access on shards!
- $\Rightarrow$ Data distribution is poorly done



Second shard results in two-shards config (49.1% of data)

The exact issue

- Mismatch between the scanning table vs data stored on disk

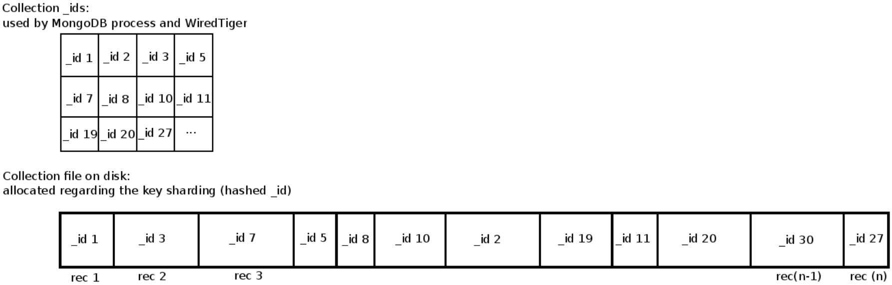


FIGURE – MongoDB scanning table Vs records order on the disk

The exact issue

- Mismatch between the scanning table vs data stored on disk

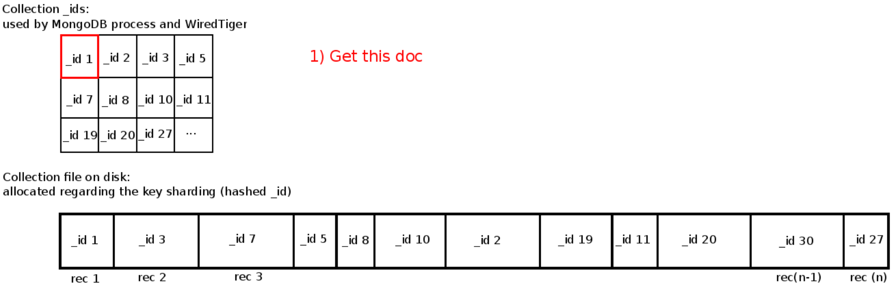


FIGURE – MongoDB scanning table Vs records order on the disk

The exact issue

- Mismatch between the scanning table vs data stored on disk

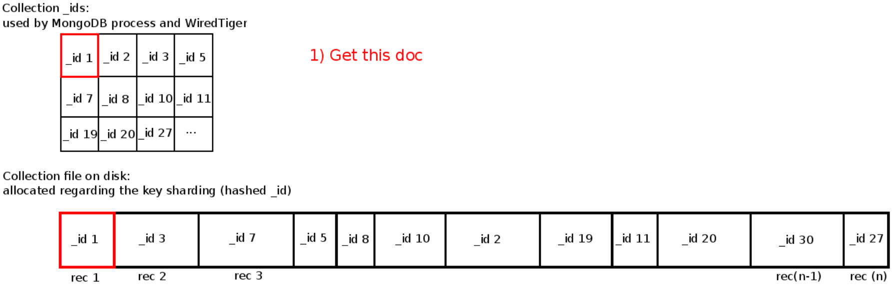


FIGURE – MongoDB scanning table Vs records order on the disk

The exact issue

- Mismatch between the scanning table vs data stored on disk

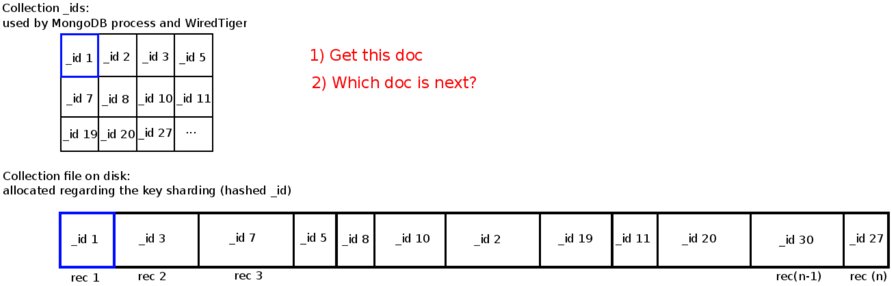


FIGURE – MongoDB scanning table Vs records order on the disk

The exact issue

- Mismatch between the scanning table vs data stored on disk

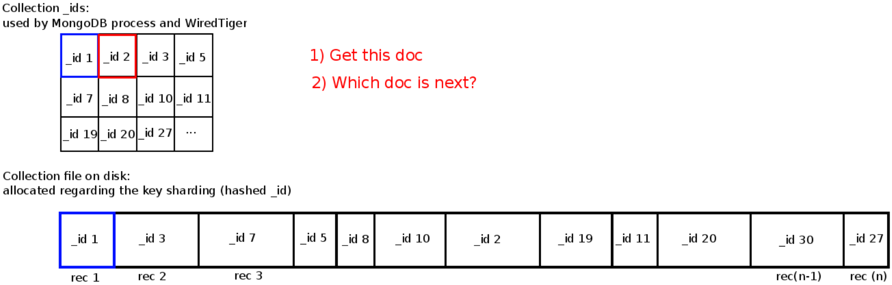


FIGURE – MongoDB scanning table Vs records order on the disk

The exact issue

- Mismatch between the scanning table vs data stored on disk

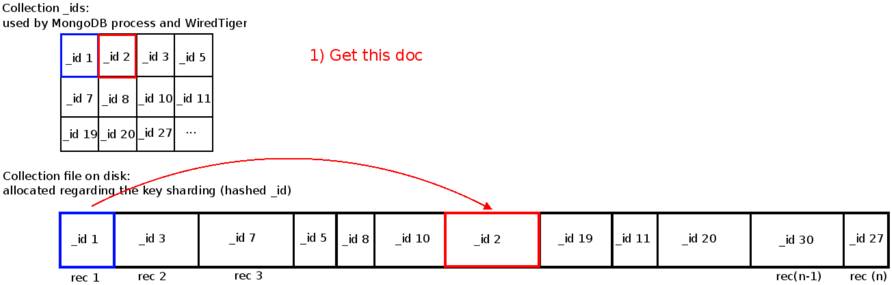


FIGURE – MongoDB scanning table Vs records order on the disk

The exact issue

- Mismatch between the scanning table vs data stored on disk

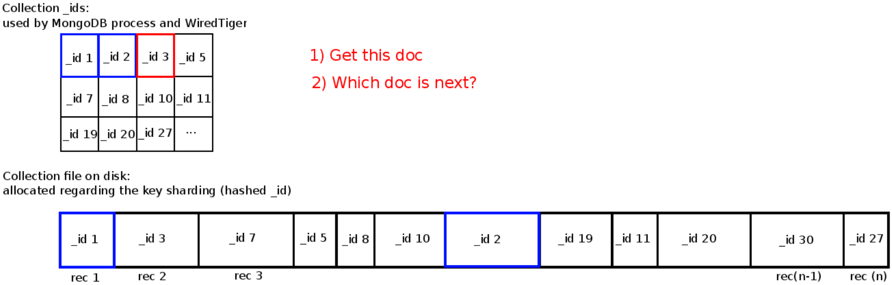


FIGURE – MongoDB scanning table Vs records order on the disk

The exact issue

- Mismatch between the scanning table vs data stored on disk

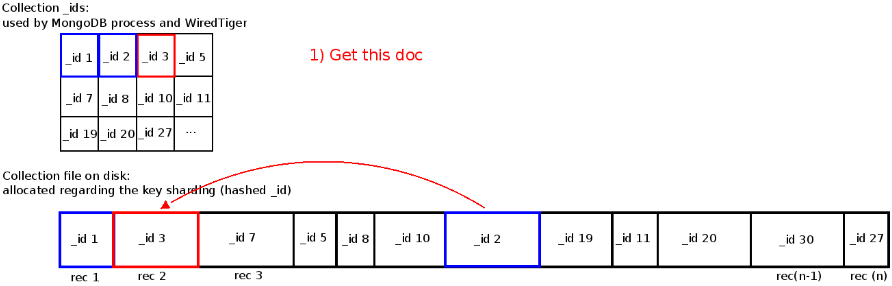


FIGURE – MongoDB scanning table Vs records order on the disk

The exact issue

- Mismatch between the scanning table vs data stored on disk

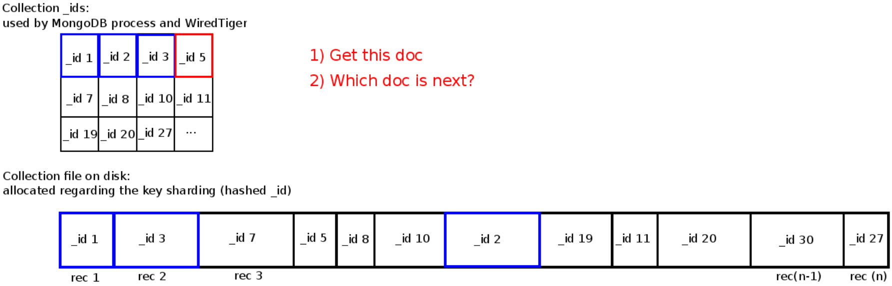


FIGURE – MongoDB scanning table Vs records order on the disk

The exact issue

- Mismatch between the scanning table vs data stored on disk

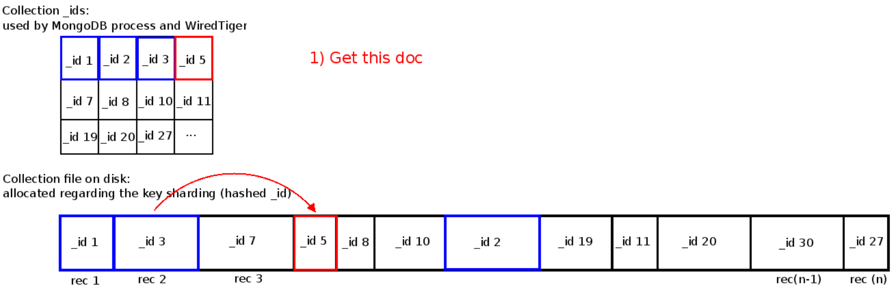


FIGURE – MongoDB scanning table Vs records order on the disk

The exact issue

- Mismatch between the scanning table vs data stored on disk

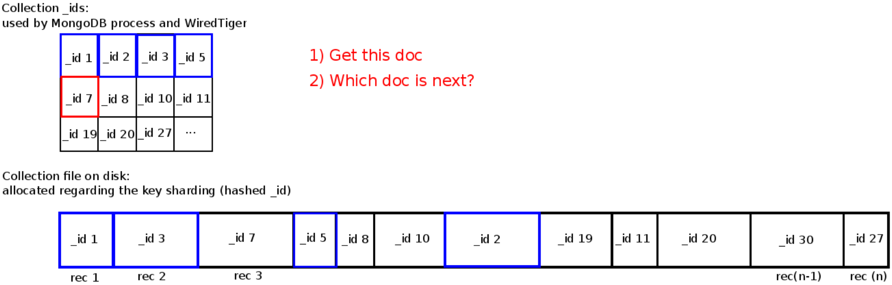


FIGURE – MongoDB scanning table Vs records order on the disk

The exact issue

- Mismatch between the scanning table vs data stored on disk

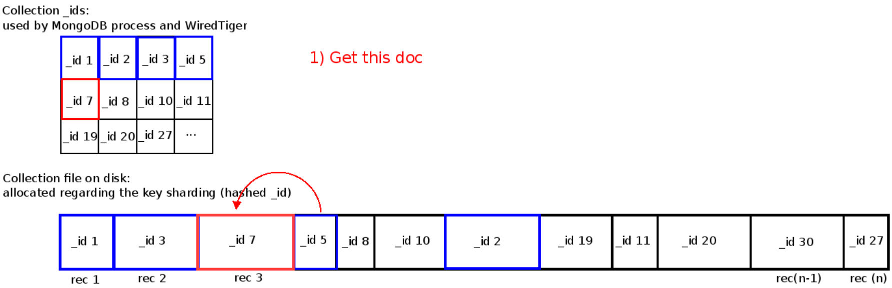


FIGURE – MongoDB scanning table Vs records order on the disk

The exact issue

- Mismatch between the scanning table vs data stored on disk

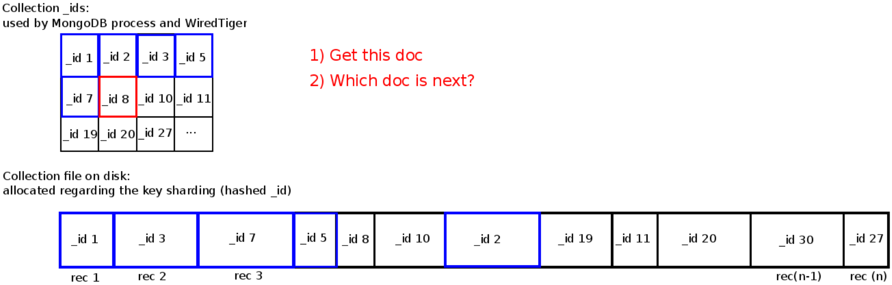


FIGURE – MongoDB scanning table Vs records order on the disk

The exact issue

- Mismatch between the scanning table vs data stored on disk

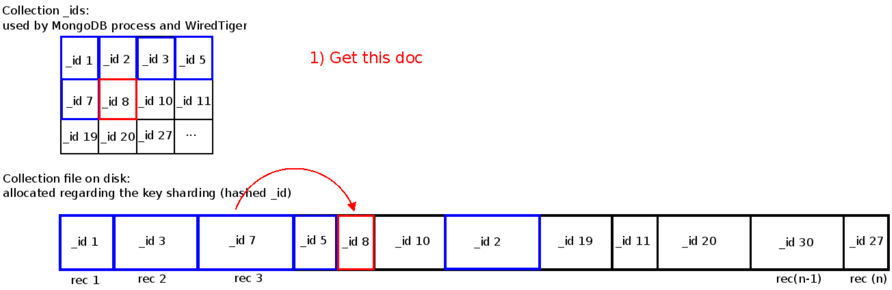


FIGURE – MongoDB scanning table Vs records order on the disk

The exact issue

- Mismatch between the scanning table vs data stored on disk

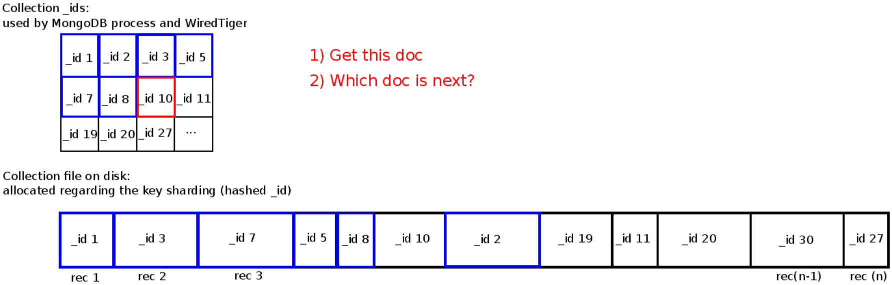


FIGURE – MongoDB scanning table Vs records order on the disk

The exact issue

- Mismatch between the scanning table vs data stored on disk

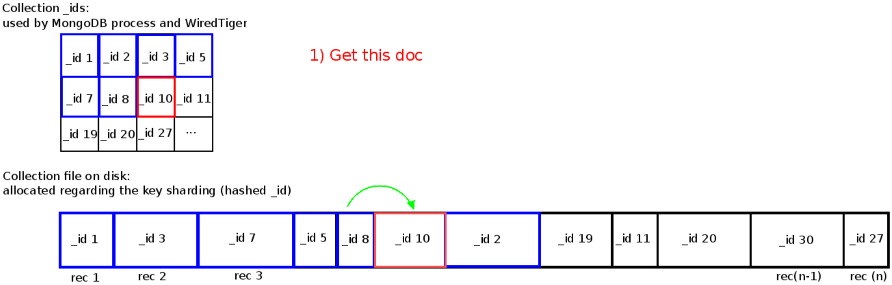


FIGURE – MongoDB scanning table Vs records order on the disk

The exact issue

- Mismatch between the scanning table vs data stored on disk

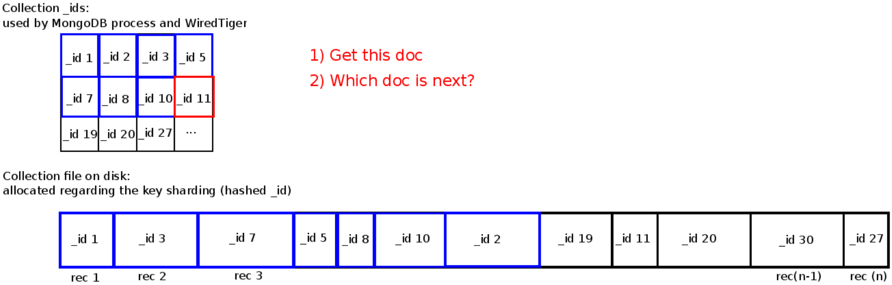


FIGURE – MongoDB scanning table Vs records order on the disk

The exact issue

- Mismatch between the scanning table vs data stored on disk

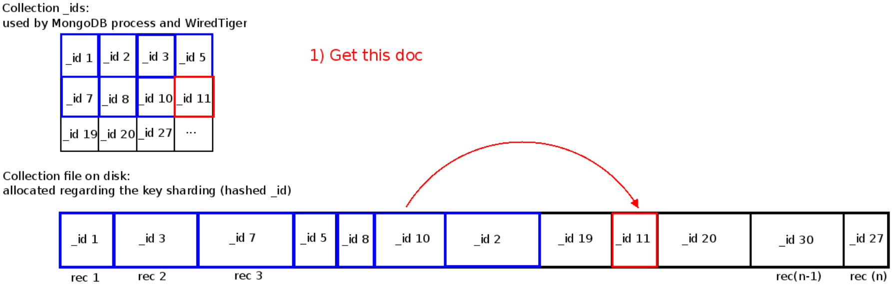


FIGURE – MongoDB scanning table Vs records order on the disk

The exact issue

- Mismatch between the scanning table vs data stored on disk

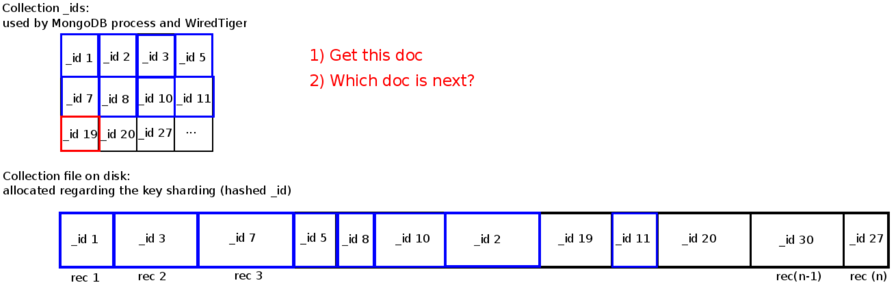


FIGURE – MongoDB scanning table Vs records order on the disk

The exact issue

- Mismatch between the scanning table vs data stored on disk

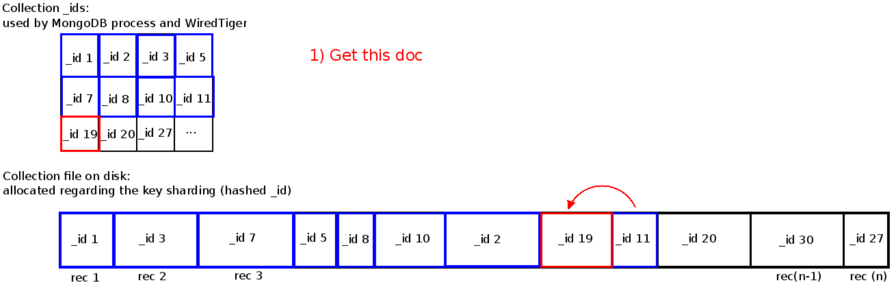


FIGURE – MongoDB scanning table Vs records order on the disk

The exact issue

- Mismatch between the scanning table vs data stored on disk

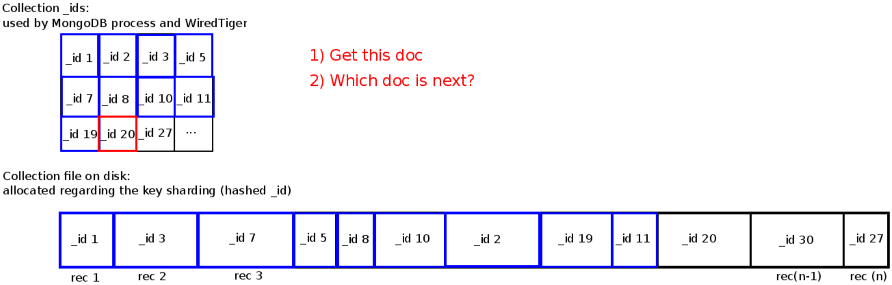


FIGURE – MongoDB scanning table Vs records order on the disk

The exact issue

- Mismatch between the scanning table vs data stored on disk

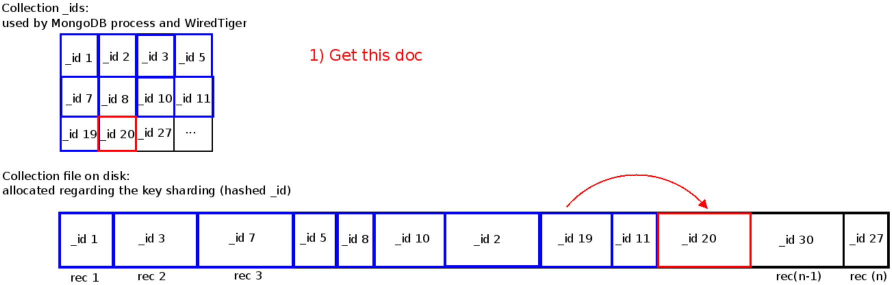


FIGURE – MongoDB scanning table Vs records order on the disk

The exact issue

- Mismatch between the scanning table vs data stored on disk

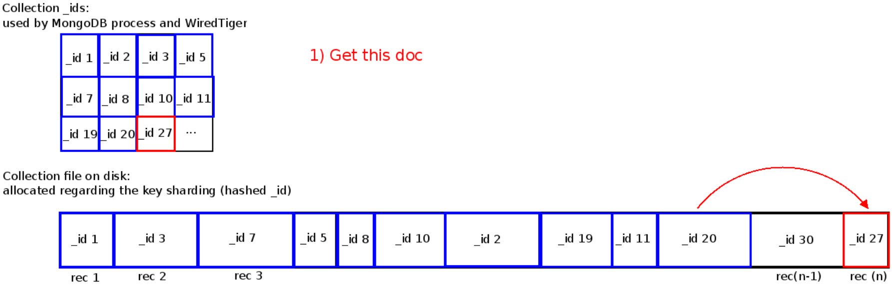


FIGURE – MongoDB scanning table Vs records order on the disk

The exact issue

- Mismatch between the scanning table vs data stored on disk

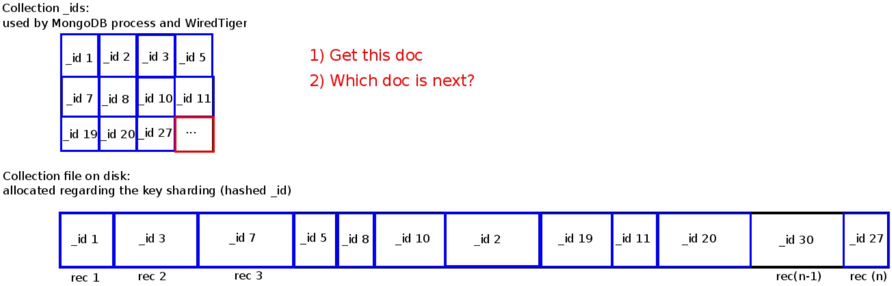


FIGURE – MongoDB scanning table Vs records order on the disk

Experiments : an ad hoc solution

MongoDB integrated tools?

- RepairDatabase ⇒ verifying the coherence of data. ☹
- Rebuilding the _id index ⇒ consider the pre-existed ordering. ☹

Experiments : an ad hoc solution

MongoDB integrated tools?

- RepairDatabase ⇒ verifying the coherence of data. ☹
- Rebuilding the _id index ⇒ consider the pre-existed ordering. ☹

Our ad-hoc solution

- recreating the _id index regarding how the data is stored on disk.

Experiments : an ad hoc solution

MongoDB integrated tools?

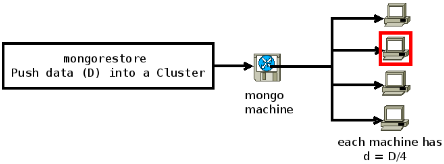
- RepairDatabase ⇒ verifying the coherence of data. ☹
- Rebuilding the _id index ⇒ consider the pre-existed ordering. ☹

Our ad-hoc solution

- recreating the _id index regarding how the data is stored on disk.

The worst case example!

- Data on one shard in a four conf. experiment (1/4 out of data)
- indexing time : Doubling the indexing time of the whole data on standalone config (3000 sec).
- indexing time after : six time faster! (about 500 sec)



Experiments : an ad hoc solution

MongoDB integrated tools?

- RepairDatabase ⇒ verifying the coherence of data. ☹
- Rebuilding the _id index ⇒ consider the pre-existed ordering. ☹

Our ad-hoc solution

- recreating the _id index regarding how the data is stored on disk.

The worst case example!

- Data on one shard in a four conf. experiment (1/4 out of data)
- indexing time : Doubling the indexing time of the whole data on standalone config (3000 sec).
- indexing time after : six time faster! (about 500 sec)

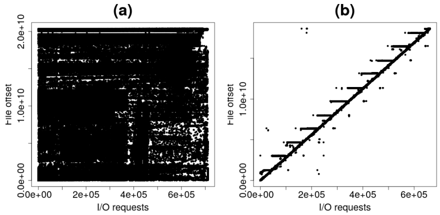


FIGURE – I/O access pattern a) before & b) after applying the solution

Conclusion

Conclusion

- A performance study on MongoDB I/O access pattern is done
- A generic tool for testing I/O access patterns is introduced
- An ad hoc solution is proposed to correct MongoDB I/O access patterns

- We showed how a trivial data management issue could affect the performance
- We demonstrated how it is worthy to use dynamic tracing to go beyond benchmarking results

Future works

- Including other solutions like Cassandra in those tests.
- Trying to enrich eBPF tools which explain specific issues.

Conclusion

Conclusion

- A performance study on MongoDB I/O access pattern is done
- A generic tool for testing I/O access patterns is introduced
- An ad hoc solution is proposed to correct MongoDB I/O access patterns

- We showed how a trivial data management issue could affect the performance
- We demonstrated how it is worthy to use dynamic tracing to go beyond benchmarking results

Future works

- Including other solutions like Cassandra in those tests.
- Trying to enrich eBPF tools which explain specific issues.

Questions are welcome !